Introdução ao Scilab 3.0
Parte 3

Paulo S. Motta Pires
pmotta@dca.ufrn.br

Departamento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte
NATAL - RN

Endereços e Créditos

Prof. Paulo S. Motta Pires

- e-mail: pmotta@dca.ufrn.br
- homepage : `www.dca.ufrn.br/~pmotta`

Agenda

- Parte 1
  - Introdução: Computação Numérica
  - O Ambiente Scilab
- Parte 2
  - Operações Básicas
  - Polinômios, Vetores, Matrizes e Listas
- Parte 3
  - Programação
- Parte 4
  - Gráficos
  - Considerações Finais

Agenda Parte 3 - Detalhes

- Introdução
- Comandos para Iterações
  - for, while
- Comandos Condicionais
  - if-then-else, select-case
- Scripts
  - Exemplo: Método de Newton-Raphson
- Funções
  - Exemplo 1: Método de Runge-Kutta
  - Exemplo 2: Método da Substituição Reversa

Características da Linguagem Scilab

- Desenvolver Programas para Computação Numérica
- Variáveis: sem declaração prévia
- Interpretada
- Portável
- Ligação com C ou FORTRAN
- Arquivos: diary, scripts, funções

Iterações com for

```
for variavel = vetor_linha
        instrucao_1
        instrucao_2
         ...   ...
        instrucao_n
end
```

Linha de Comando: Iterações com for

```
-->for variavel=vetor_linha
-->   instrucao_1
-->   instrucao_2
-->   instrucao_n
-->end
```

```
--> Loop for em linha de comando

-->for k = 1:3
-->a = k + 1
-->end
 a  =

    2.
 a  =

    3.
 a  =

    4.

-->
```

Linha de Comando: Iterações com for

```
-->v = [2 3 4 5 6]; // Pode ser  v = 2:6

-->y = 0;

-->for k = v
-->y = y + k
-->end
 y  =

    2.
 y  =

    5.
 y  =

    9.
 y  =

    14.
 y  =

    20.

-->
```

```
-->L = list(1, [1 2; 3 4], 'teste')
 L  =


       L(1)

    1.

       L(2)

!   1.    2. !
!   3.    4. !

       L(3)

 teste

-->for k=L
-->disp(k)
-->end

    1.

!   1.    2. !
!   3.    4. !

 teste
```

Iterações com while

```
while condicao
      instrucao_1
      instrucao_2
       ...   ...
      instrucao_n
 end
```

Operadores	Significado
== ou =	igual a
<	menor do que
>	maior do que
<=	menor ou igual a
>=	maior ou igual a
<> ou ~=	diferente

Tabela: Operadores condicionais

Linha de Comando: Iterações com while

```
-->while condicao
-->   instrucao_1
-->   instrucao_2
-->   instrucao_n
-->end
```

```
-->x = 1;

-->while x < 14
-->x = x * 2
-->end
 x  =

    2.
 x  =

    4.
 x  =

    8.
 x  =

    16.

-->
```

Comandos Condicionais - if-then-else

```
if condicao_1 then
     sequencia_de_instrucoes_1
else
     sequencia_de_instrucoes_2
end
```

ou

```
if condicao_1 then
     sequencia_de_instrucoes_1
elseif condicao_2
     sequencia_de_instrucoes_2
   ... ... ... ... ... ...
elseif condicao_n
     sequencia_de_instrucoes_n
else
     sequencia_de_instrucoes_n+1
end
```

Comandos Condicionais - if-then-else

```
--> if condicao_1 then
-->    sequencia_de_instrucoes_1
-->elseif condicao_2
-->    sequencia_de_instrucoes_2
--> elseif condicao_n
-->    sequencia_de_instrucoes_n
--> else
-->    sequencia_de_instrucoes_n+1
-->end
```

```
-->x = -1;

-->if x < 0 then
-->   y = -x          // apresenta a resposta
 y  =

    1.
-->else
-->   y = x
-->end

-->// Outra forma

-->x = 1          // Inicializando
 x  =

    1.

-->if x > 0 then, y = -x, else, y=x, end
 y  =

  - 1.

-->x = -1
 x  =

  - 1.
```

Comandos Condicionais - select-case

```
select variavel_de_teste
case expressao_1
        sequencia_de_instrucoes_1
case expressao_2
        sequencia_de_instrucoes_2
  ... ... ... ... ... ... ... ...
case expressao_n
        sequencia_de_instrucoes_n
else
        sequencia_de_instrucoes_n+1
end
```

Comandos Condicionais - select-case

```
-->select variavel_de_teste
-->case expressao_1
-->   sequencia_de_instrucoes_1
-->case expressao_2
-->   sequencia_de_instrucoes_2
-->case expressao_n
-->   sequencia_de_instrucoes_n
-->else
-->   sequencia_de_instrucoes_n+1
-->end
```

```
-->x = -1
 x  =

  - 1.

-->select x
-->case 1
-->   y = x + 5
-->case -1
-->   y = sqrt(x)
 y  =

    i
-->end

-->x = 1
 x  =

   1.

-->select x,case 1,y = x+5,case -1,y = sqrt(x),end
 y  =

    6

-->
```

Scripts

- Contém comandos Scilab
- Texto puro (ASCII)
- Convenção - extensão sce

Para executar:

```
-->exec(‘‘nome_do_arquivo_de_comandos.sce’’)
```

Scripts - Observações

- Todas as variáveis definidas no arquivo de comandos permanecem válidas no ambiente Scilab após a execução dos comandos do arquivo, e
- Não há uma definição clara das entradas e saídas do *script*. Esse fato pode dificultar a correção de possíveis erros.

Scripts - Exemplo

Script na linguagem Scilab para obter a $\sqrt{2}$ usando o método iterativo de Newton-Raphson. A raiz de uma função, $f(x)$ pode ser calculada através da expressão,

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

onde $f'(x_i)$ representa a derivada da função $f(x)$ no ponto x_i e $i = 0, 1, \ldots, n$ representa o número de iterações.
A função $f(x)$ que permite obter a $\sqrt{2}$ é

$$f(x) = x^2 - 2$$

Assim, usando a fórmula de Newton-Raphson, temos

$$x_{i+1} = x_i - \frac{x^2 - 2}{2x}$$

Scripts - Código

Usando $x_0 = 1.0$ como aproximação inicial

```
// Script: metodo de Newton-Raphson
// f(x) = x * x - 2

N = 10;      // Numero maximo de iteracoes
x0 = 1.0;    // Aproximacao inicial
delta = 10^(-5);  // Erro

xn = x0;
for n=1:N
    xn1 = xn - (xn * xn - 2)/(2 * xn);
    if abs((xn1-xn)/xn1) < delta then
       printf('Valor da raiz = %10.7f',xn1)
       return
    end
        xn = xn1;
end
printf('Nao converge em n=%f iteracoes", N)
```

Observar o ; após alguns comandos. Isso evita que a execução do *script* seja um processo "ruidoso" no ambiente Scilab.

Scripts - Execução

```
-->exec newton.sce      // executando o script

-->// script: metodo de Newton-Raphson

-->// f(x) = x * x - 2


-->N = 10;      // Numero maximo de iteracoes

-->x0 = 1.0;    // Aproximacao inicial

-->delta = 10^(-5);  // Erro


-->xn = x0;

-->for n=1:N
-->    xn1 = xn - (xn * xn - 2)/(2 * xn);
-->    if abs((xn1-xn)/xn1) < delta then
-->       printf('Valor da raiz = %10.8f',xn1)
-->       return
-->    end
-->        xn = xn1;
-->end
Valor da raiz = 1.41421356       // valor de sqrt(2) !
-->
```

Scripts - Comentários

```
Valor da raiz =  1.4142136      // Final da execucao
-->// Variaveis permanecem ativas no ambiente Scilab

-->N
 N  =

    10.

-->x0
 x0  =

    1.

-->xn1
 xn1  =

    1.4142136

-->delta
 delta  =

    0.00001

-->
```

Funções - Forma Geral

```
function [y1, ..., yn] = foo(x1, ..., xm)
     instrucao_1
     instrucao_2
        ...
     instrucao_p
endfunction
```

onde **foo** é o nome da função, **xi**, i=1,...,m, são os seus argumentos de entrada, **yj**, j=1,...,n, são os seus argumentos de saída e **instrucao_i**, i=1,...,p, representa a seqüência de instruções que devem ser executadas pela função.
Toda função no Scilab é executada chamado seu nome seguido de seus argumentos.No exemplo, a função **foo** é executada através do comando

```
-->foo(x1, ..., xm)
```

Funções - Considerações

- As variáveis definidas na função, chamadas de variáveis locais, não permanecem no ambiente Scilab após a execução da função
- As entradas e saídas do programa são claramente definidas
- Uma função, após ser definida, pode ser chamada a qualquer tempo

Funções - Criação

- Digitação no próprio ambiente

```
-->Digitando uma funcao no ambiente Scilab

-->function [y1, y2] = exemplo(x1, x2)
-->// Entrada: x1, x2
-->// Saida: y1, y2
-->y1 = x1 + x2
-->y2 = x1 * x2
-->endfunction

-->[a,b] = exemplo(2, 3)
 b  =

    6.
 a  =

    5.

-->
```

Funções - Criação

- Usando o comando deff

```
-->Usando deff

-->deff('[y1, y2]=exemplo(x1, x2)','y1=x1+x2, y2=x1*x2')

-->[a, b] = exemplo(3,4)
 b  =

    12.
 a  =

    7.

-->
```

Funções - Criação

- Digitando o texto da função em um arquivo e, em seguida, carregando esse arquivo no ambiente Scilab. Por convenção, as funções definidas pelo usuário possuem extensão sci e são carregadas no ambiente Scilab através do comando:

```
-->getf(‘‘nome_do_arquivo_de_comandos.sci’’)
```

Arquivos com Funções

É possível definir funções dentro do próprio ambiente Scilab. Entretanto, quando a função possui muitos comandos, é mais conveniente utilizar um editor de textos para criar, fora do ambiente Scilab, um arquivo contendo a função. É importante observar que o nome desse arquivo não é, necessariamente, o nome que deve ser dado à função.

Funções - Exemplo Runge-Kutta

Equação diferencial ordinária de primeira ordem:

$$\frac{dy}{dx} = f(x, y)$$

Runge-Kutta de quarta ordem:

$$y_{k+1} = y_k + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4)$$

com os coeficientes f_i definidos por :

$$\begin{aligned} f_1 &= f(x_k, y_k) \\ f_2 &= f(x_k + \frac{h}{2}, y_k + \frac{h}{2} f_1) \\ f_3 &= f(x_k + \frac{h}{2}, y_k + \frac{h}{2} f_2) \\ f_4 &= f(x_k + h, y_k + h f_3) \end{aligned}$$

Algoritmo Runge-Kutta

Entrada: $[a, b]$, h e $y0$
Fazer $x0 = a$
Fazer $n = (b - a)/h$
for $k = 0$ *to* n **do**
  Calcular f_1, f_2, f_3 e f_4
  Calcular $y_{k+1} = y_k + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4)$
  Fazer $x_{k+1} = x_k + h$
end
Resultado: Apresentar valores de x_k e y_k

Runge-Kutta - Exemplo

Resolver a equação diferencial ordinária,

$$\frac{dy}{dx} = (x - y)/2$$

com condição inicial $y(0) = 1$, utilizando o método de Runge-Kutta de quarta ordem. Vamos considerar o intervalo de integração $[0, 3]$ e o passo de integração igual a $h = 1/8$. A solução numérica obtida por Runge-Kutta será comparada com valores da solução exata que é $y(x) = 3e^{-x/2} + x - 2$.

Função Principal

rk4.sci

```
function [XY] =  rk4(a, b, h, y0)
// Resolucao de equacoes diferenciais ordinarias por Runge-Kutta 4a. ordem
// Entrada : [a,b] - intervalo de integracao
//                h - passo da integracao
//                          y0 - condicao inicial em x0
X(1) = a
Y(1) = y0
Exato(1) = f2(X(1), Y(1))
n = (b-a)/h
for k=1:n
        xk = X(k)
        yk = Y(k)
        hf1 = h * f(xk, yk)
        hf2 = h * f(xk + h/2, yk + hf1/2)
        hf3 = h * f(xk + h/2, yk + hf2/2)
        hf4 = h * f(xk + h, yk + hf3)
        Y(k+1) = Y(k) + (hf1 + 2*hf2 + 2*hf3 + hf4)/6
        X(k+1) = X(k) + h
        Exato(k+1) = f2(X(k+1), Y(k+1))
end
XY = [X Y Exato]
endfunction
```

Função rk4.sci - Editor

SciPad - /home/paulo/metodos/sciport-1.5/rkutta4.sci

File Edit Search Windows Options Execute Help

```
function [XY] =  rk4(a, b, h, y0)
// Resolucao de equacoes diferenciais ordinarias por Runge-Kutta 4a. ordem
// Entrada : [a,b] - intervalo de integracao
//                h - passo da integracao
//               y0 - condicao inicial em x0
X(1) = a
Y(1) = y0
Exato(1) = f2(X(1), Y(1))
n = (b-a)/h
for k=1:n
  xk = X(k)
  yk = Y(k)
  hf1 = h * f(xk, yk)
  hf2 = h * f(xk + h/2, yk + hf1/2)
  hf3 = h * f(xk + h/2, yk + hf2/2)
  hf4 = h * f(xk + h, yk + hf3)
  Y(k+1) = Y(k) + (hf1 + 2*hf2 + 2*hf3 + hf4)/6
  X(k+1) = X(k) + h
  Exato(k+1) = f2(X(k+1), Y(k+1))
end
XY = [X Y Exato]
endfunction
```

Line: 15 Column: 3 | Line 15 in rk4

Funções Auxiliares

fdexy.sci

```
function [fxy] = f(x,y)
// funcao exemplo
fxy = (x - y)/2
endfunction
```

fsolucao.sci

```
function [fexato] = f2(x,y)
// funcao solucao
fexato = 3 * exp(-x/2) + x - 2
endfunction
```

Funções - Ambiente Scilab

```
-->getf('fdexy.sci')     // arquivo com a funcao f(x,y) = (x - y)/2

-->getf('fsolucao.sci')  // arquivo com a funcao solucao = 3exp(-x/2)+x-2

-->getf('rkutta4.sci')   // arquivo com o metodo de Runge-Kutta 4a. ordem

-->rk4(0, 3, 1/8, 1)     // executando o programa

 ans  =

!   0.        1.           1.         !
!   0.125     0.9432392    0.9432392  !
!   0.25      0.8974908    0.8974907  !
!   0.375     0.8620874    0.8620874  !
!   0.5       0.8364024    0.8364023  !
!   0.625     0.8198470    0.8198469  !
!   0.75      0.8118679    0.8118678  !
!   0.875     0.8119457    0.8119456  !
!   1.        0.8195921    0.8195920  !
!   1.125     0.8343486    0.8343485  !
!   1.25      0.8557844    0.8557843  !
!   1.375     0.8834949    0.8834947  !
!   1.5       0.9170998    0.9170997  !
!   1.625     0.9562421    0.9562419  !
!   1.75      1.0005862    1.0005861  !
```

Funções - Exemplo Substituição Reversa

Sistema triangular superior:

$$\begin{array}{rcrcrcrcr} 4x_1 & - & x_2 & + & 2x_3 & + & 2x_4 & - & x_5 & = & 4 \\ & - & 2x_2 & + & 6x_3 & + & 2x_4 & + & 7x_5 & = & 0 \\ & & & & x_3 & - & x_4 & - & 2x_5 & = & 3 \\ & & & & & - & 2x_4 & - & x_5 & = & 10 \\ & & & & & & & & 3x_5 & = & 6 \end{array}$$

Matriz dos coeficientes aumentada:

$$[A \mid \mathbf{b}] = \left[\begin{array}{ccccc|c} 4 & -1 & 2 & 2 & -1 & 4 \\ 0 & -2 & 6 & 2 & 7 & 0 \\ 0 & 0 & 1 & -1 & -2 & 3 \\ 0 & 0 & 0 & -2 & -1 & 10 \\ 0 & 0 & 0 & 0 & 3 & 6 \end{array}\right]$$

Algoritmo - Substituição Reversa

$$x_n = \frac{b_n}{a_{n,n}}$$

for $r = n - 1$ *até* 1 **do**

$soma = 0$

for $j = r + 1$ *até* n **do**

$soma = soma + a_{r,j} * x_j$

end

$$x_r = \frac{b(r) - soma}{a_{r,r}}$$

end

Funções - Código Substituição Reversa

progtri.sci

```
function X = subst(Tsup)
// Entrada : matriz triangular superior, Tsup
// Saida : Vetor solucao X
[nl, nc] = size(Tsup);
n = nc-1;
A = Tsup(:,1:n);  // Matriz A
b = Tsup(:,nc)    // Vetor  b
X = zeros(n,1);
X(n) = b(n)/A(n,n);
for r = n-1:-1:1,
        soma = 0,
        for j = r+1:n,
                soma = soma + A(r,j) * X(j);
        end,
X(r) = (b(r) - soma)/A(r,r);
end
endfunction
```

Funções - Exemplo Substituição Reversa

```
paulo:~/metodos/funcoes/aula3$ cat arquivo1

4 -1  2  2 -1  4
0 -2  6  2  7  0
0  0  1 -1 -2  3
0  0  0 -2 -1 10
0  0  0  0  3  6

-->Ab = read('arquivo1', 5, 6)
 Ab  =

!   4.   - 1.    2.    2.   - 1.    4.  !
!   0.   - 2.    6.    2.    7.    0.  !
!   0.    0.    1.   - 1.   - 2.    3.  !
!   0.    0.    0.   - 2.   - 1.    10. !
!   0.    0.    0.    0.    3.    6.  !

-->
```

Funções - Exemplo Substituição Reversa

```
-->[nl nc] = size(Ab)  // nl - linhas, nc - colunas
 nc  =

    6.
 nl  =

    5.

-->A = Ab(:,1:nc-1)   // Matriz dos coeficientes
 A  =

!   4.  - 1.    2.    2.  - 1. !
!   0.  - 2.    6.    2.    7. !
!   0.    0.    1.  - 1.  - 2. !
!   0.    0.    0.  - 2.  - 1. !
!   0.    0.    0.    0.    3. !

-->b = Ab(:,nc)       // Vetor dos termos independentes
 b  =

!   4.  !
!   0.  !
!   3.  !
!   10. !
!   6.  !
```

Funções - Exemplo Substituição Reversa

Solução

```
-->getf('progtri.sci')   // Arquivo com o metodo da Substituicao Reversa

-->subst(Ab)
 ans  =

!   5. !
!   4. !
!   1. !
! - 6. !
!   2. !

-->
```